International Journal of Electrical and
Electronics Engineering Research (IJEEER)
ISSN(P): 2250-155X; ISSN(E): 2278-943X
Vol. 5, Issue 2, Apr 2015, 11-18

TRANS STELLAR
• Journal Publications • Research Consultancy

AMBA- ADVANCED HIGH PERFORMANCE BUS MASTER BASED VERIFYING INTELLECTUAL PROPERTY USING OPEN VERIFICATION METHODOLOGY

PALAKEETI NAVEEN KALYAN[1] & B MALLESWARI[2]

[1]Student, Department of ECE, QIS College of Engineering and Technology, Ongole, Andhra Pradesh, India

[2]Associate Professor, Department of ECE, QIS College of Engineering and Technology, Ongole, Andhra Pradesh, India

ABSTRACT

The Advanced Microcontroller Bus Architecture (AMBA) specification defines an on-chip communications standard mainly used for the purpose of designing high-performance embedded M Microcontrollers. AMBA was introduced by ARM Ltd in 1996, and nowadays, it is widely used as the on-chip bus in SOC designs. In this paper, a verification environment to verify an ARM-based SOC (System-on-Chip) by using open verification methodology (OVM) is presented. The paper also introduced how to verify the AMBA (Advanced Microprocessors Bus Architecture) by open verification methodology, include AHB (Advanced High Performance Bus) Master where AHB is a part of the AMBA protocol. Verification technology is designed to enable creation of robust, interoperable verification IP and test-bench components to verify any AMBA protocol based SoC it improves quality and reduces schedule time it is the standard framework to build the verification environment waveforms, code coverage is also discussed in the paper.

KEYWORDS: SOC (System on Chip), AHB (Advance High Performance Bus), AMBA (Advanced Microcontroller Bus Architecture), OVM (Open Verification Methodology)

INTRODUCTION

In, now a days, the new advancements of VLSI technology enables the integration of more than several million transistors in a single chip to make a SOC(System-on-Chip). Generally, the task of designing and verifying the IP was done by two engineers namely, design engineer and verification engineer. The main and the huge work was done by the verification engineer. Today, testing as a word has been replaced with the verification. The main aim of the verification is the finding of bugs, you only partly correct. The goal of hardware design is to create a device that performs a particular task, such as a DVD player, network router, or radar signal processor, based on a design specification.

Your purpose as a verification engineer is to make sure the device can accomplish that task successfully – that is, the design is an accurate representation of the specification. So, verification engineers have to ensure what goes to the factory for manufacturing is an accurate representation of the design specification. But verifying the IP is a more complex task than designing IP. So, for the purpose of the verification, in this paper, we used a methodology called as open verification methodology which is an advanced version of system Verilog. OVM is a complete verification methodology that codifies the best practices for development of verification environments targeted at verifying large gate-count, IP-based SoCs.

Verification productivity stems from the ability to quickly develop individual verification components, encapsulatethem into larger reusable verification components (OVCs), and reuse them in different configurationsand at different levels of abstraction. So, OVM is used for the purpose of verification of semi-conductor chip designs. There are

two methodologies namely, constrained random stimulus and functional coverage which are used for the purpose of verification. Here, functional coverage functional verification comprises a large portion of the resources required to design and validate a complex system. Often, the validation must be comprehensive without redundant effort. In current industrial practice, simulation-based verification is the main functional verification for large and complex designs. With dynamic verification, the verification is done by generating a massive number of tests using random test generators, simulating the tests on the design, and checking that the design behaves according to its specification. Coverage checks completeness of the Testbench. This can be improved by the usage of Assertions, which helps to check the coverage of the Testbench and generate suitable reports of the test coverage.

The concept of coverage can get more complex, when we deal with the concept of functional coverage, cover groups and cover points. With the coverage points, we can generate coverage report of your design and know the strength of your verification. Using OVM has test case multiple tests are compiled together and a test case can be chosen at the run time using command line the constraint random test cases in the OVM are much shorter than the directed test cases with OVM test class we don't need to recompile the environment again to run a new test case which saves huge time for big verification environments. In this paper, how to put up the simulation environment using Open Verification Methodology is introduced. The simulation environment using by OVM is efficient and high-performance. The paper discusses the usage experience to verify a SoC by the IP and the AHB master during verify the SoC, a great deal of visual simulation waveform inspection is required. To generate the waveform will be time-consuming. The paper presents SoC bus transaction verification IP written by Open Verification Methodology, which are called reference model to compare the behaviour of the DUT. The IP can be integrated into OVM simulation environment. We can greatly reduce the waveform inspection time.

OPEN VERIFICATION METHODOLOGY ENVIRONMENT

The purpose of the OVM test bench is to determine the correctness of the design under test. This is going to be done as firstly generating stimulus, after applying that generated stimulus to the design under test then after capturing the response and finally checking the results to verify that the function is correct. Then test cases may be modified or added referring to the coverage reports. By using Open Verification Methodology we can easily do these works.

Introduction of the Verification Environment

Figure 1 shows the block diagram of the OVM verification environment. It includes the DUT which was written by Verilog's test benches. The test bench consists of OVM_agent, OVM _driver, OVM_monitor, OVM_sequencer. The working of the test bench is as follows. Firstly, starting from the generator. The generator is used for the purpose of creating constrained random test vectors. After that all these test vectors are sent to the driver then the driver starts the process of stimulating the DUT. From the driver now the signal information is sent to the monitor. The monitor extracts signal information from a bus and translatesthe information into a transaction that can be made available to other components and to the test writer.

All this information is made available to other components using transaction level modelling interface. The monitor collects bus information through a virtual interface. The collected data is used in coverage collection and checking for scoreboard. Figure 2 shows the diagram of OVM_agent. The OVM test bench can configure the agent as either active or passive. In the active mode the agent will start the first phase of OVM, i.e., the agent will start the process

of building upon the verification components like sequencer, driver, monitor. In the active configuration active agent will emulate the devices and drive transactions according to test directives. In the Passive mode, the agent consists of only the monitor, sequencer, driver, but are not included the agent inside. The passive agent only monitor DUT activity, and it doesn't drive stimulus into the DUT the interface part (Interface) is the interface to be used to joining the DUT and the test program. The verification environment will also be reused, without modifications, by as many test cases as possible to minimize the amount of codes required to verify the DUT.

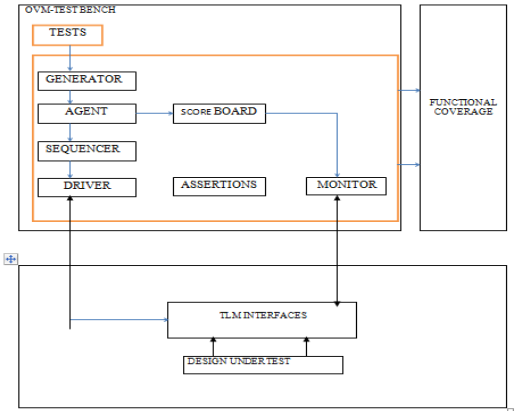

Figure 1: Open Verification Methodology Environment Block Diagram

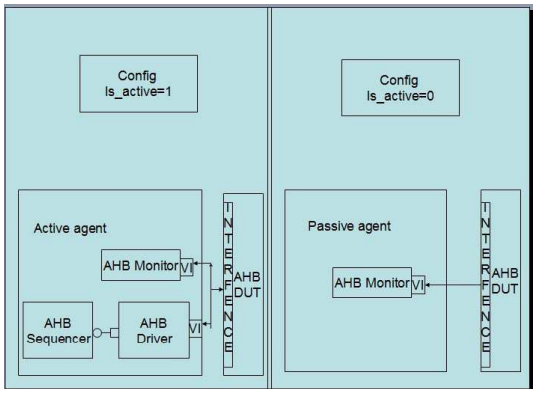

Figure 2: OVM Agent Background

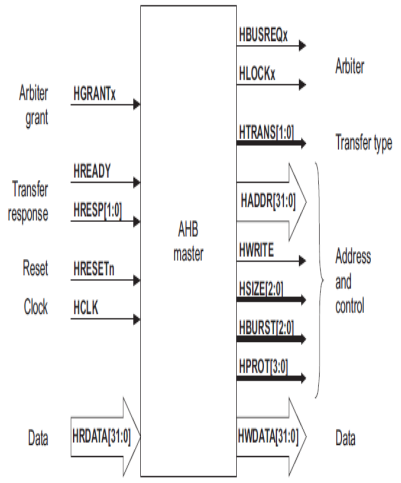

Figure 3: AHB Master Block Diagram

AMBA AHB Master Design

Figure 3: shows the AHB master block diagram. The AHB bus will have the four components namely master, slave, arbiter, decoder. As referring to the figure 3 AHB master block diagram, the signals HBUSREQx, HTRANS, HWDATA, HSIZE, HLOCK,

HADDR are the master signals and the HRDATA, HRESP, HREADY are the slave signals. Starting from HBUSREQx the master will send a request signal to the arbiter (i.e to the HGRANT signals) to get the granted access to the bus. Then the arbiter indicates when the master will granted uses the bus. After getting the access then the AHB transfer can commence, and the AHB bus also has provisions of several masters but only one master must be access for the data transfer and that master only can access the slaves at a given time. A granted bus master starts an AMBA AHB transfer by driving the address and control signals. Here the data can transfer in the form of four ways as idle, busy, non-sequential, Sequential.

The bus master will uses the HTRANS signal to what type of data is going to be transfer and by using HWDATA signal the bus master can transfer the data from master to the slave. Only one master can access the slaves at the given time. All of the other masters must respect this and wait until the bus is assigned to them, and the slave will use the HRDATA signal to send the response to the master signal and the slave can drive HREADY low to stretch the length of a bus cycle. The master must observe the status of the HRESP. If RETRY or SPLIT is given, the master must immediately drive the IDLE value on its HTRANS output. When the HLOCK is asserted high indicates that the master requires locked access to the bus and no other master should be granted the bus until this signal is LOW. The main feature of the AHB is that. In fact, the address phase of any transfer occurs during the data phase of the previous transfer. This overlapping of address and data is fundamental to the pipelined nature of the bus and allows for high performance operation.

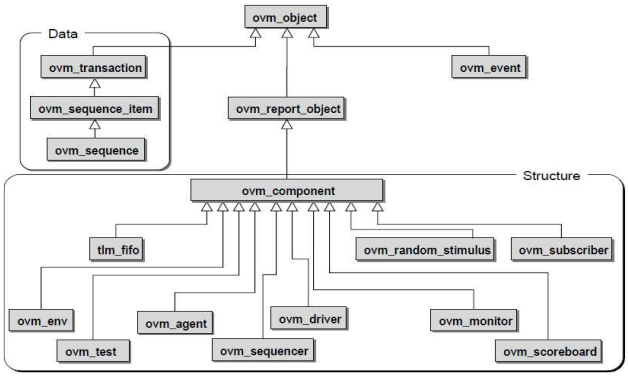

Figure 4: Execution Flow in the OVM Verification

OPEN VERIFATION METHODOLOGY HIERARCHY

OVM is announced in the year 2007, by cadence design system and mentor graphics as a joint effort to provide a common methodology to verification. By using of OVM we can write number of test cases, independent of test bench environment. As shown in the figure 4 executing flow in the OVM hierarchy. It consists of OVM components, OVM test. OVM component classes are sequencer, driver, monitor, scoreboard.

OVM_ Object: Is the virtual base class for all components and transactions in an OVM environment.

OVM_ Sequencer: It is a verification component that mediates the generation and flow of data between

sequences and a driver. The sequencer has a collection of sequences associated with it called a sequence library. The collection of sequences used by a sequencer is called sequence library. This type of component is also referred to as a driver sequencer.

OVM_ Generator: It is used to generate the test vectors.

OVM-Driver: generator create inputs at high level of abstraction namely as transactions like read write operation. The driver convert this input signal into actual design inputs, as defined in the specification of the design interfaces. A driver is a verification component that connects at the pin-level interface to the DUT. It contains one or more transaction-level interfaces to communicate with other transaction-level components in the verification environment.

OVM_ Monitor: Monitor reports the protocol violation and identifies all the transactions. Monitors are two types, Passive and active. Passive monitors do not drive any signals. Active monitors can drive the DUT signals. Sometimes this is also referred as receiver. Monitor converts the state of the design and its outputs to a transaction abstraction level so it can be stored in a 'score-boards' database to be checked later on. Monitor converts the pin level activities in to high level.

OVM_ Agent: A device that contains the standard components necessary to drive HDL signals (the driver), provide stimulus to the driver (the sequencer) and collect data items along with enforcing checks and tabulating coverage (the monitor). An agent is capable of independent operation.

OVM_Scoreboard: A scoreboard predicts the response of the design dynamically. The stimulus applied to the design is provided to a transfer function. It is the transfer function that performs all transformation operations on the stimulus in order to produce the form of the final response then inserts it in a data structure. Score board will have two analysis ports. One is used for getting packets from the driver and other from the receiver

Transaction: it means basically what needs to be tested. Examples of transaction are instruction, data, parameters, where instruction represents the high-level tasks to be executed, such as a READ, WRITE, NO-OP, LOAD, etc. The data represents such as address, data, number of cycles, etc, and the parameters represents a mode, a size, etc.

Interface: interface is the mechanism to connect Testbench to the DUT just named as bundle of wires (e.g. connecting two hardware unit blocks with the help of physical wires)

AHB MASTER BEHAVIOUR ANALYSIS

AHB master is first configured with different configuration parameters. The master then initiates requests based on read and write commands from the testbench. These commands include low level commands and burst commands. For burst commands, a new command buffer is created and the burst transfer attributes are set using rich set of buffer commands. Read data and response from slave are stored in response buffer. User can access this buffer to get read data or slave response. For the AHB master environment, the different tasks used in driver by a master are RESET, READ, WRITE, RUN. In the reset task, the reset operation will be taking place, making HRESETn LOW for 3 clock signal and the Read and Write operation take place When HRESETn HIGH. In the read task, the read operation will be taking place, Here we will be reading the HRDATA,HRESP, HREADY, HGRANT signals coming from slave. In the write task, Write operation will take place this will only be performed when HGRANT signal is HIGH, which is based on the Grant process by Arbiter,

In the run task will call RESET, READ, WRITE Task in a fork join any.

Coverage is defined as the percentage of verification objectives that have been met. It is used as a metric for evaluating the progress of a verification project in order to reduce the number of simulation cycles spent in verifying a design Broadly speaking, there are two types of coverage metrics. Those that can be automatically extracted from the design code, such as code coverage, and those that are user-specified in order to tie the verification environment to the design intent or functionality. Functional coverage is a user-defined metric that measures how much of the design specification, as enumerated by features in the test plan, has been exercised.

It can be used to measure whether interesting scenarios, corner cases, specification invariants, or other applicable design conditions— captured as features of the test plan—have been observed, validated and tested.

The key aspects of functional coverage are. It is user-specified, and is not automatically inferred from the design It is based on the design specification (i.e., its intent) and is thus independent of the actual design code or its structure. It individual simulation generates a data base of functional coverage information. By merging all this information together, overall progress can be measured by using functional coverage this information is only valid for a successful simulation as shown in figure 5 showing the simulation result of coverage with cover groups. We achieved the 100percent of functional coverage for AHB master.

Covergroups / Instances	Total Bins	Covered Bins (Hits)	% Hit	Goal	Coverage	% of Goal
Covergroup cover_signal_in	11	11	100.00%	100.00%	100.00%	**100.00%**
Instance \ahb_top/ahb_if /#ublk#108955126#87/cg	11	11	100.00%	100.00%	100.00%	**100.00%**

Covergroup type:cover_signal_in					100.00%	100.00%	**100.00%**
Coverpoints / Bins	**At Least**	**Total Bins**	**Covered Bins (Hits)**	**% Hit**	**Goal**	**Coverage**	**% of Goal**
Coverpoint: haddr_cover		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: dst_addr_cover		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: src_addr_cover		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: datain_cover		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: hrdata_cover		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: hwdata_cover		1	1	100.00%	100.00%	100.00%	100.00%
Coverpoint: dataout_cover		1	1	100.00%	100.00%	100.00%	100.00%
Cross: cross_dst_Haddr <haddr_cover,dst_addr_cover>		1	1	100.00%	100.00%	100.00%	100.00%
Cross: cross_src_Haddr <haddr_cover,src_addr_cover>		1	1	100.00%	100.00%	100.00%	100.00%
Cross: cross_readdata <hrdata_cover,dataout_cover>		1	1	100.00%	100.00%	100.00%	100.00%
Cross: cross_writedata <hwdata_cover,datain_cover>		1	1	100.00%	100.00%	100.00%	100.00%

Figure 5: Shows Functional Coverage

AHB Master Wave form Analysis

Burst operation is defined as one or more data transactions, initiated by a bus master, which have a consistent width of transaction to an incremental region of address space. The increment step per transaction is determined by the width of transfer (byte, halfword, word).

Generally, we are having two types of BURSTS namely wrapping burst and Incrementing burst. Incrementing bursts access sequential locations and the address of each transfer in the burst is just an increment of the previous address. Now here the size of transfer is given as “011” which means an amount of the 64 bits transfer, and the type of the burst “011”represents the burst size as INCR4 means 4-beat incrementing burst and hence four continues write or read operations happens. Here the count is introduced in order to generate the address with respect the given initial address and the count is increment. The operation remains the same as simple read and write but the only change is that after each operation, count will check for the burst size given. If the count of the burst is not equal to the size of the burst given it means for example count of burst is 16 bit then the burst size must also be 16 bit, if both are not equal, then the count will be incremented and of course we are taken the transfer size as “011” (INCR4).

So then the count will be incremented by 4 bits it mean showing as an example as if the count size is 0x38 then it will be incremented with 4 bits as 0x3C and again with 4 bits as 0x40. This will done like this, the count will get incrementedand the next address is get generated based on which the read or write operation that currently performed is carried out. When the count is equal to burst length, that represents the burst operation over and count resets to zero. Hence master and slave go IDLE state.

Table 1: Transfer Type of AHB Master

00	IDLE
01	BUSY
10	NON SEQ
11	SEQ

HTRANSFER operation starts at non-sequential and it is followed by sequential the address is related to the previous transfer (see table 1) that are four different types IDLE where the IDLE indicates that no data transfer is required. The IDLE transfer type is used when a bus master is granted the bus, but does not wish to perform a data transfer. The next transfer busy indicates the BUSY transfer type allows bus masters to insert IDLE cycles in the middle of bursts of transfers. This transfer type indicates that the bus master is continuing with a burst of transfers, but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst.

The non-sequential indicates the first transfer of a burst or a single transfer. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of one and therefore the transfer type is NONSEQUENTIAL. Sequential indicates the remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. HWRITE signal will be asserted as high when the HBURST write operation taking its place and the HWRITE signal will be asserted as low when the HBURSTread operation taking place and the HWRITE will made as don't care operation after the burst operation is over which is shown in Figure 6. Similarly, HREADY signal asserted as high which indicates the transfer has finished on the bus which is shown in Figure 7.

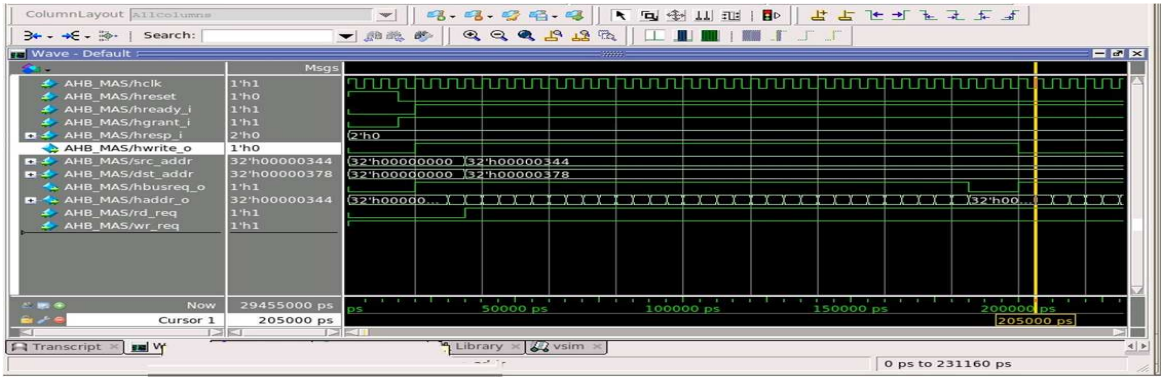

Figure 6: Wave Form for AHB Master-Write Operation

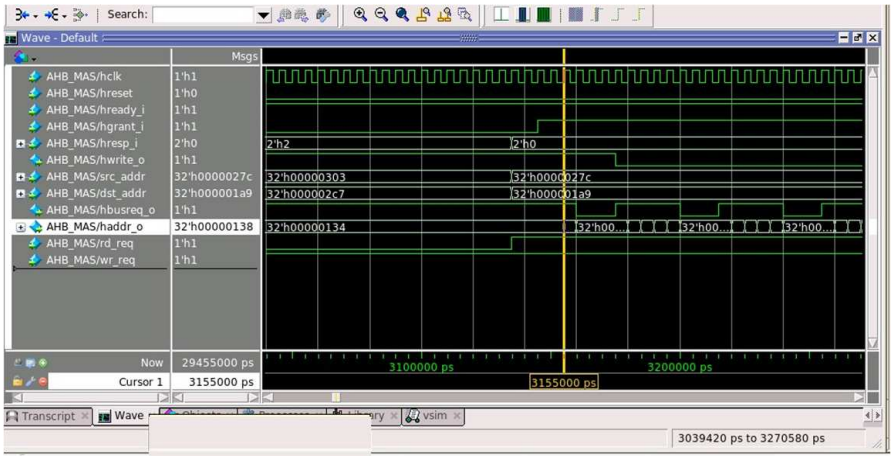

Figure 7: Wave form for AHB Master-Increment Burst Operation (INCR4)

CONCLUSIONS

In this paper we use open verification methodology (OVM) to put a verification environment this methodology reduces the simulation time required to find bugs, when compared to a conventional methodology. The checker keeps track of all the activity during simulation and checks the response right way we analyse the functional coverage with cover groups, cover points and bins we achieve the 100% functional coverage for AHB master.

ACKNOWLEDGEMENTS

We thank B. Malleswari for explaining the changes made in the specifications we are grateful to yashdeep for many insightful comments about our specifications that helped us improve our paper.

REFERENCES

1. ARM Ltd. AMBA Specification Rev.2.0 May 1999.http://www.arm.com
2. OVM User Guide, Version 2.1.1, March 2010 http://www.ovmworld.org
3. Open Verification Methodology: Fulfilling the Promise of System Verilog' by Thomas L. Anderson, Product Marketing Director Cadence Design Systems, Inc.
4. ‘System Verilog for Verification: A Guide to Learning the Test bench Language Features’ by Chris Spear s. l.: Springer, 2006
5. Chris spear, system Verilog for verification, New York: springer, 2006.